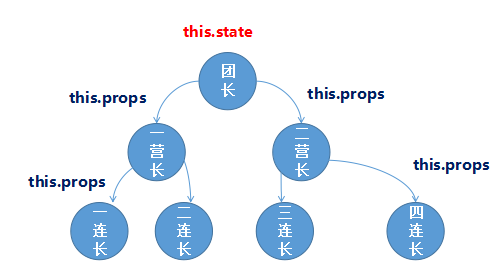
# [redux和react-redux小记](https://www.cnblogs.com/bax-life/p/8440326.html)

一开始接触redux时对于它的数据state和react本身的数据state还有点容易打结（它们是完全不同的两个东西，后面会讲到）现在搞清楚一点了，来小结一下。

在此之前，还是先讲一下redux是干嘛的，也就是说它是解决什么需求的，这里举一个小例子。

先来看看React里面的数据是怎样传递的（**下图中并不是说this.state只能在“团长”处，这里只是为了简便，所以才这样写，其实营长，连长都可以有自己的this.state**）：

**React数据流：**



**1.**可以看到，一般都是团长等高级干部商量好对策之后（this.state设置好）

**2**. 把命令一层层的传递下去执行（this.props渲染）

**3**. 要是哪里发生了什么新的敌情了（想更新页面内容了）

**4**.就得一层层往上报告（通过回调一层层把数据传上去），此过程也必须一层层往上传递，不能越级，下面有解释

**5**. 团长做出决断后再发布命令传下去执行（通过setState修改数据并重新渲染）

**这个过程是不能越级的**，比如说你一连长发现了敌情就不能直接报告团长，而必须先报告给一营长，一营长再报告给团长，这里层级还不是特别深，你再来个排长，班长什么的，按照这种模式，等团长知道敌情，敌人都打到家门口啦，又或者说一连长想找三连长喝个酒，还得先把这个消息传给营长，再传给团长，再传给二营长再传给三连长，这……

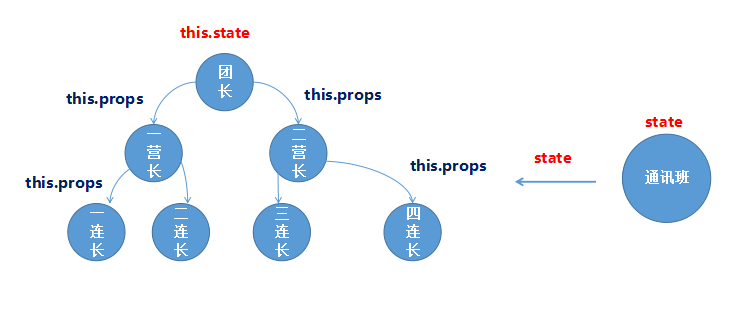
**Redux横空出世**

所以说层级不深，组价之间也没有什么公用数据的时候，用React自身的setState其实也可以了，可是嵌套一深，或者组件有一些公用数据时就比较麻烦了，于是就有了redux（其实之前还有个flux），回到这个例子就是，他们在整个团之外又建立了一个**通讯班（Redux）**：

**分清两个state**

在说Redux之前，我想说一下自己学的时候遇到的一个小坑，就是**Redux中的state和React中的state完全不是一回事**，React中的state是组件内部自己的状态信息，而Redux中的state是Redux自己的数据，然后React就拿Redux中的数据来用，其实Redux也可以在其他框架下使用，并不是非要跟React一起使用。

简单画个图看起来就像下面这样：



这里只是一个很简单的示意图，实际使用并非如此，这个图示为了让大家理解React中的数据和Redux中的数据是独立的，并没有半毛钱关系

**Redux**

现在让我们抛开战斗编制（React），单看通讯班（Redux）。

其实网上讲Redux的教程非常多，这里就简单说一下：

### **store**

store:首先要创建一个对象store，这个对象有各种方法，用来让外界获取Redux的数据（store.getState），或者让外界来修改Redux中的数据（store.dispatch）

import { createStore } from 'redux';

const store = createStore(reducer);

### **action**

action:描述我要干啥，一般是一个对象的形式，其中有一个type字段是必须要有的，比如：{ type：‘请求增援’ }，还可以带点数据{ type：‘请求增援’，gun:"100" }

### **reducer**

reducer:撸开袖子真刀实枪的就去干了，比如一连长要求增援，增援要求是100杆枪，团长马上就给你加了100杆枪送了过去。

[IMG_258](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

const defaultState = 0;

const reducer = (state = defaultState, action) => {

switch (action.type) {

case '请求增援':

return state + action.gun;

default:

return state;

}

};

[IMG_259](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

action和reducer也可以想象成产品经理和程序员的关系。

产品经理：“我要一个按钮，圆角的”

程序员：“嗯，做好了”

产品经理：“换个颜色吧，红色”

程序员：“ok，换好了”

差不多就是这样，产品经理并不操心具体怎么实现的，他只说他想要干什么（type），然后再提点实现的要求（各种其他的数据），程序员就去具体实现（reducer，修改state，然后返回一个**全新的state,**这里注意我们并没有直接返回原来的state，我们返回的是一个全新的state对象，因为reducer函数是一个没有副作用的纯函数）

那怎么去触发这个操作（action）呢，就好比说我一连长发现敌情了，我怎么报告给通讯班，让通讯班来处理呢？

就是上面提到过的store.dispatch，不过还要加一个参数，那就是action

store.dispatch({ type：‘请求增援’，gun:"100" })

这样就可以触发action，执行reducer，得到一个全新的state。

## **Redux 和 React**

到此为止，Redux自己就折腾完了，那么Redux自己的数据并没有用，它要把数据交给React用才行，接下来讲一讲怎么把数据交给React来用。

上面我们创建了一个对象store，我们要把这个store对象作为props传给React，那React就可以用了。

这个store只能有一个，也就只能创建一次，也就是说你必须在最顶层处创建一个store对象，然后再一层层的传递下去，才能让所有的组件都能获得这个store对象，调用它的方法。

### **获取Redux中的数据**

比如说我要在render函数中显示Redux的数据，那么我就可以先获取它的数据：

store.getState()

然后把这个数据当做props渲染到组件中去就行了。

### **更新Redux中的数据**

如果你要修改它的数据，那就在JSX中调用

store.dispatch( { type：‘请求增援’，gun:"100" } )

### ****响应Redux中的变化****

那么这里问题又来了，你调用了store.dispatch之后Redux中的数据确实改变了，可是React并没有什么变化啊。也就是说React中的render函数并没有被触发呀是不是，就好像React中你直接修改React中的state是没有用的而必须调用React的setState才能重新渲染。

因此，为了让Redux的数据一改变我们就重新渲染，Redux自己提供了一个方法叫做

store.subscribe(render)

这个函数可以**监听Redux中state的变化**，一旦Redux中的state发生了变化，render函数就会被调用，页面就会被重新渲染。

上面这个过程就是手动调用的过程，但这样调用有点麻烦，因为要让所有的子组件都能应用store中的数据，那么**所有的组件就都要把store当做props传进来**，这也太麻烦了。

还是那之前那个例子来说，你一连长发现了敌情，是不用层层上报了，可以直接报告给通讯班，通讯班再生成新的命令，可问题是，你从下往上打报告是简单了，可是从上往下发布命令仍然是一层层的传递的呀。

就好比说，一连长发现了敌情，报告上去，通讯班做出决定让三连长带人去打，通讯班还是得通过团长--营长--连长这条路去一层层发布命令，能不能让通讯班直接就通知三连长呢，当然是有的，这就是我们的React-redux库

## **React-redux**

这个是需要你自己去用npm额外安装的。

使用这个方法之后，我们就不需要一层层往下发布命令了

在React-redux中有两个比较关键的概念：Provider和connect方法。

一般我们都将顶层组件包裹在Provider组件之中，这样的话，所有组件就都可以在react-redux的控制之下了,**但是store必须作为参数放到Provider组件中去**

<Provider store = {store}>

<团长/>

<Provider>

这个组件的目的是让所有组件都能够访问到Redux中的数据。

 这个比较简单，我们主要讲connect方法。

### **connect方法：**

connect(mapStateToProps, mapDispatchToProps)(MyComponent)

其实connect方法一共有4个参数，这里主要讲前两个。

#### **mapStateToProps**

字面含义是把state映射到props中去，意思就是把Redux中的数据映射到React中的props中去。

也就是说你React想把Redux中的哪些数据拿过来用。

比如这里二连这个组件想要渲染自己枪支的数量。就可以直接在二连这个组件中把Redux中的gunOfErlian拿过来用

const mapStateToProps = (state) => {

return {

gun: state.gunOfErlian

}

}

然后渲染的时候就可以直接使用this.props.gun

[IMG_260](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

class Erlian extends Component {

constructor(props){

super(props);

}

render(){

return(

<div>this.props.gun</div> )

}

}

Erlian = connect()(Erlian);

export default Erlian;

[IMG_261](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

那么这样就可以实现渲染，就是把Redux中的state变成React中的props。

#### **mapDispatchToProps**

通过上面的分析，相信这个函数也很好理解，就是把各种dispatch也变成了props让你可以直接使用

然后就到了我们这里最重要的一点了。

[IMG_262](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

const mapDispatchToProps = (dispatch) => {

return {

onClick: () => {

dispatch({

type: '请求增援',  
　　　　 gun : 100

});

}

};

}

[IMG_263](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

更改一下上面的Erlian组件

[IMG_264](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

class Erlian extends Component {

constructor(props){

super(props);

}

render(){

return(

<div>this.props.gun</div>

<button onClick = {this.props.onClick}>请求增援</button> )

}

}

Erlian = connect()(Erlian);

export default Erlian;

[IMG_265](https://www.cnblogs.com/bax-life/p/javascript:void(0);)

当我点击请求增援按钮后，Erlian组件的枪支数量会自动的更新，而**不需要**我们手动的去用store.subscribe订阅render函数以达到更新页面的目的。

这样一来我们就不需要一层层的传递store对象了。

这种随处都可以使用、修改Redux中的数据的方式确实很方便，但Redux推荐的最佳实践还是在尽可能少的地方使用connect，把逻辑，数据相关的都放到容器组件中去处理，其他的组件都由容器组件所生成的props一层层传递下去然后渲染（傻瓜组件），这里就不多说了。

这一次的小结差不多就这样了，许多代码细节部分都没有写，更多的是对他们的作用的理解，因为自己在学习的时候，更多的是一些理解方面的困惑，而更多更全面的代码部分在Redux官方文档或是阮一峰老师的博客中都已经写得非常清楚，大家可以去看看，理解了上面许许多多的名词以及他们的含义之后，相信那些文档会更容易看懂。

上述若有错漏之处，烦请各位在评论区指出